# Team 5

## Table of Contents

# Chapter 1 Features

Features included in ***GRAPHI***:

1. *Multiple plotting on a single graph:*
   - The app has the capability to plot multiple functions in different colours
   - This enables users to directly view comparisons between functions in a given range
   - In case the functions are related or connected in any way, for example:

     $$\text{Eqn 1: } y = x$$

     $$\text{Eqn 2: } y = \sin(x)$$

     $$\text{Eqn 3: } y = \sin(x) + x$$

     The variance of function 3 can be seen as functions 1 and 2 vary over the specified range. This is especially useful to **study the functions and obtain roots of a system of equations by simple observation**

2. *Re-plot the graphs using different parameters* :
   - The graph of each function can be very easily modified by simply changing the function parameters in the script box, making it very easy for users
   - **The range of observation** of the function can also be changed using the User Interface or by typing in braces using the script box
   - **Step size used in the evaluation** can be changed in the graph settings panel, thereby varying the accuracy of plotting obtained. This enables even older systems to be used for the same plotting purposes
3. *Different features* (like colour, line types, line width) **for different function plots**. These can be the default settings or user-specified
4. *Store the graph in different file formats*:
   - For vector graphics, the app saves the plots in .svg, .pdf (for 2D plots) format
   - For raster graphics, the app saves the plots in .bmp (3D), .png & .jpeg (2D) formats
5. *Specify graph axis labelling* using the boxes in the graph settings panel
6. *Zooming, panning and orienting of 3-D plots*:

Zooming can be done by scrolling mouse wheel up and down

- By default, X and Y axes are zoomed

Holding control enables zoom function on the Z axis

        - Holding shift and control zooms the Y axis
        - Holding alt and control zooms the X axis
    - Hold down the middle mouse and move the mouse button to pan the plot
        - Horizontal motion changes the X range
        - Vertical motion changes the Y range
        - Holding down control changes the Z range
    - Plot orientation is changed by holding down the mouse left button
        - Horizontal motion changes orbital angle
        - Vertical motion changes the elevation

## 7. *Value at a point*

- Values of a point on the plot can be determined by placing the pointer at the desired point and pressing control

# Chapter 2: Architecture

## a. Use-case diagram:

The only actor in our system is the user, who is given the ability to enter functions/data series, draw and save plots.

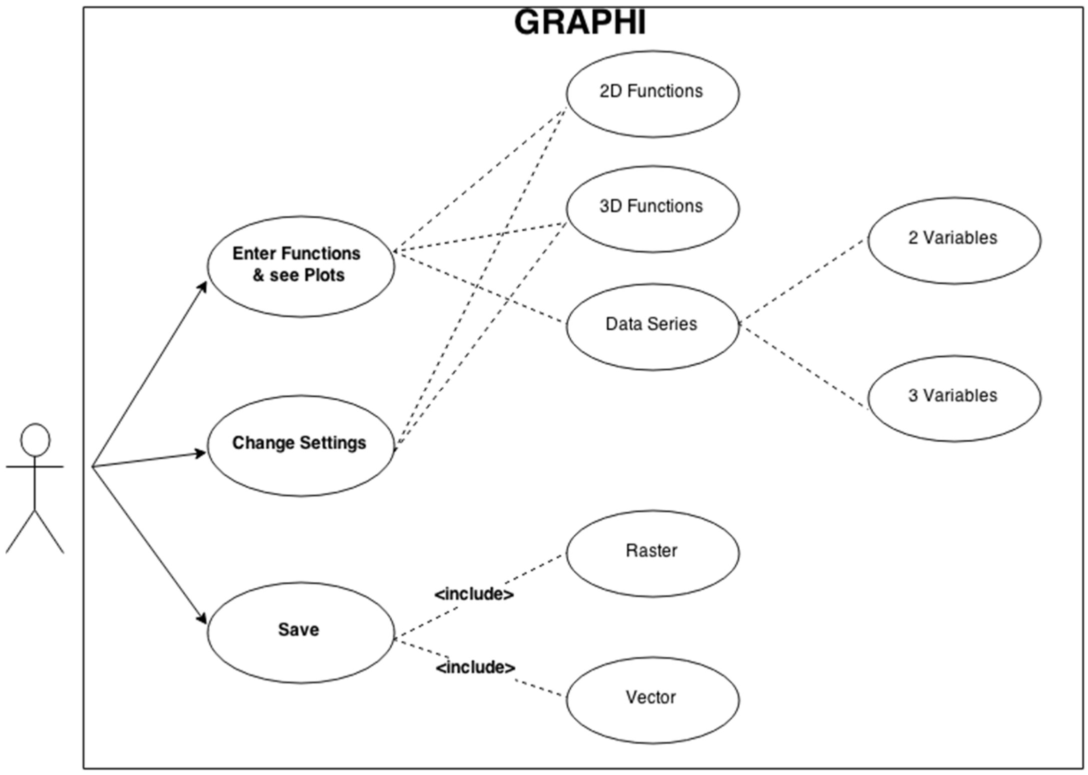

**Fig 1**: Use-case Diagram

## b. Application Modules and Interactions:

The software is divided into a clear set of modules, that interact together to produce a graph from mathematical input. We have

- A Graphical User Interface, for providing the user with an easy and intuitive way to visualise and interact with graphs.
- A Mathematical Parser, which parses, mathematical expressions into their corresponding C++ representations.
- An OpenCL based Evaluator, that builds a kernel from the C++ string and operates on a range.
- A Plotter module, that depending on the type of expression, renders a two dimensional or a three dimensional representation of the data.
- A Controller/Director, that separates the frontend from the backend and controls the flow of data across modules.
- A saving functionality.

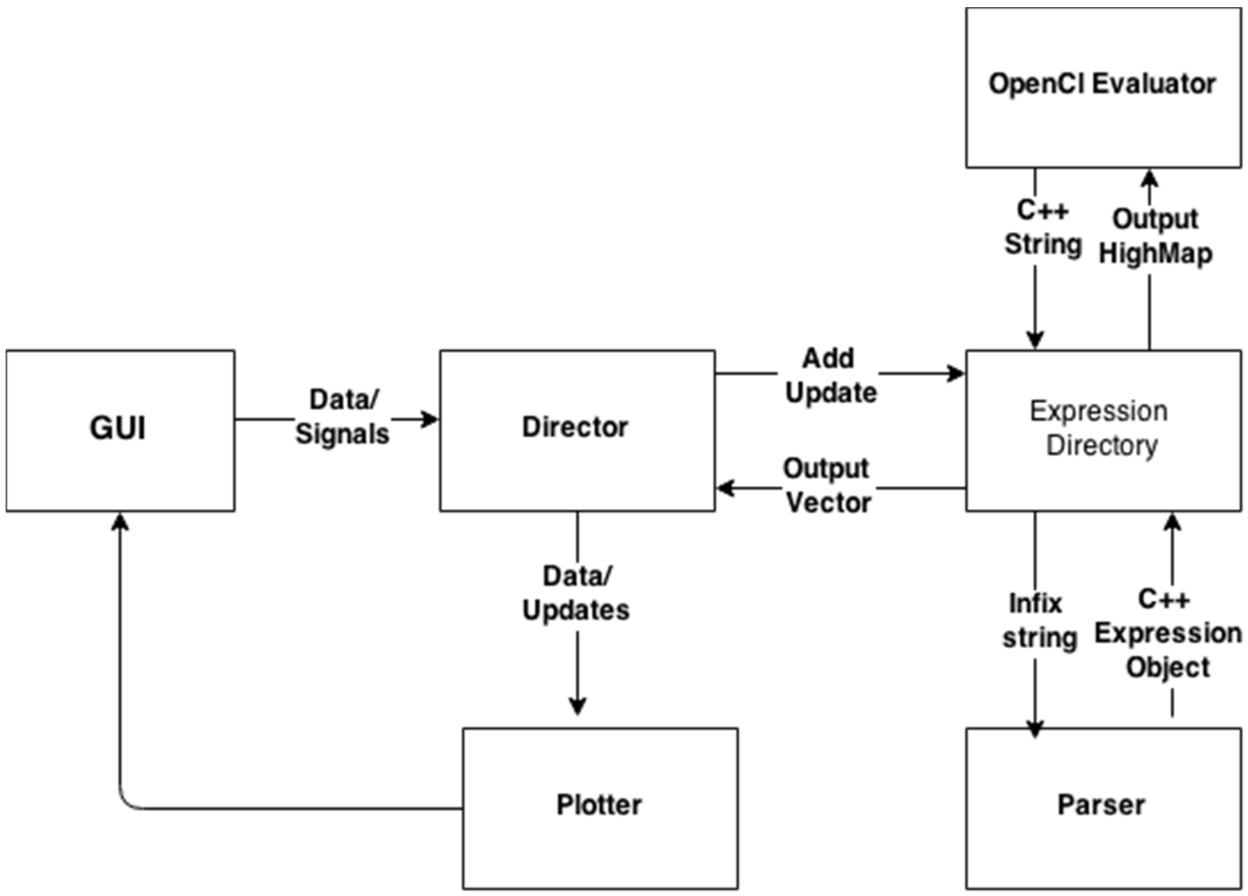

**Fig 2:** Application Module

### i. Graphical User Interface:

The user interface is very intuitive, and is primarily targeted to work with both traditional desktop PCs, and also touch screen PCs. It is designed using QtQuick, and supports dynamic interactions through both mouse and touch.

Mathematical Parser: We have used our own parser for the purpose of intelligently understanding and detecting what the user types. We have relied on OpenCL for our evaluation needs. In the Evaluator module, we utilise the equivalent C++ strings instead of direct mathematical expressions, for building our kernel. This necessitates the use of a custom "Infix To C++"parser. This Parser is able to handle explicit, implicit and parametric expressions, and automatically detects the type of the expression.

Another important feature is that the parser is able to handle implicit multiplication. Thus when the user enters xyxxyxyxyxxxx, it is not only interpreted

correctly, but also simplified to yield a string which is mathematically faster (in our example ((x^9)*(y^4)).
The Parser is also fundamental from the point that it stops all kinds of errors from user, and doesn't let them propagate through the pipeline

### ii. OpenCL Kernel:

OpenCL allows writing programs that execute across heterogeneous platforms, and allows for parallelism and access to GPUs. In our case, since we evaluate the same expression at multiple points within the range, parallelism makes for an elegant and intelligent choice.

The OpenCL based Evaluator takes in a C++ string and creates a kernel (program code) out of it. This kernel is different for each expression. We can then run this kernel, on various data points in parallel.

Utilising OpenCL in such an application has the following advantages:

- OpenCL also allows executing the code on the often underutilised Graphical Processing Units which speeds up execution as compared to using only the CPU. It utilises the hardware to its maximum extent
- It allows for massive parallel programming on huge datasets
- Threads are limited by CPU capacity, but OpenCL has no such limitations

OpenCL is device independent, and doesn't necessarily require a GPU.

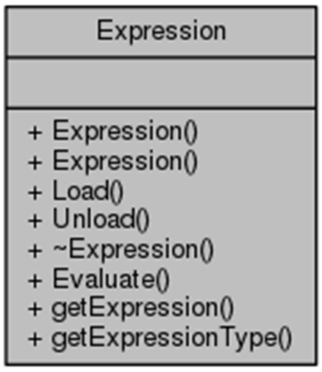

**Fig 3:** Expression Class Diagram

### iii. Base Function:

The Base function is an abstract class that manages different types of mathematical constructs like explicit (simple) mathematical constructs, implicit plots, and parametric, relation, and scatter plot and their corresponding evaluations. The base function class manages which function evaluation and plotting needs to be invoked depending on the type of expression (determined by the parser) typed. Base function also controls the settings for visualization of graphs. We have also extended the QCustomPlot, by adding the creation of contour and shading algorithms.

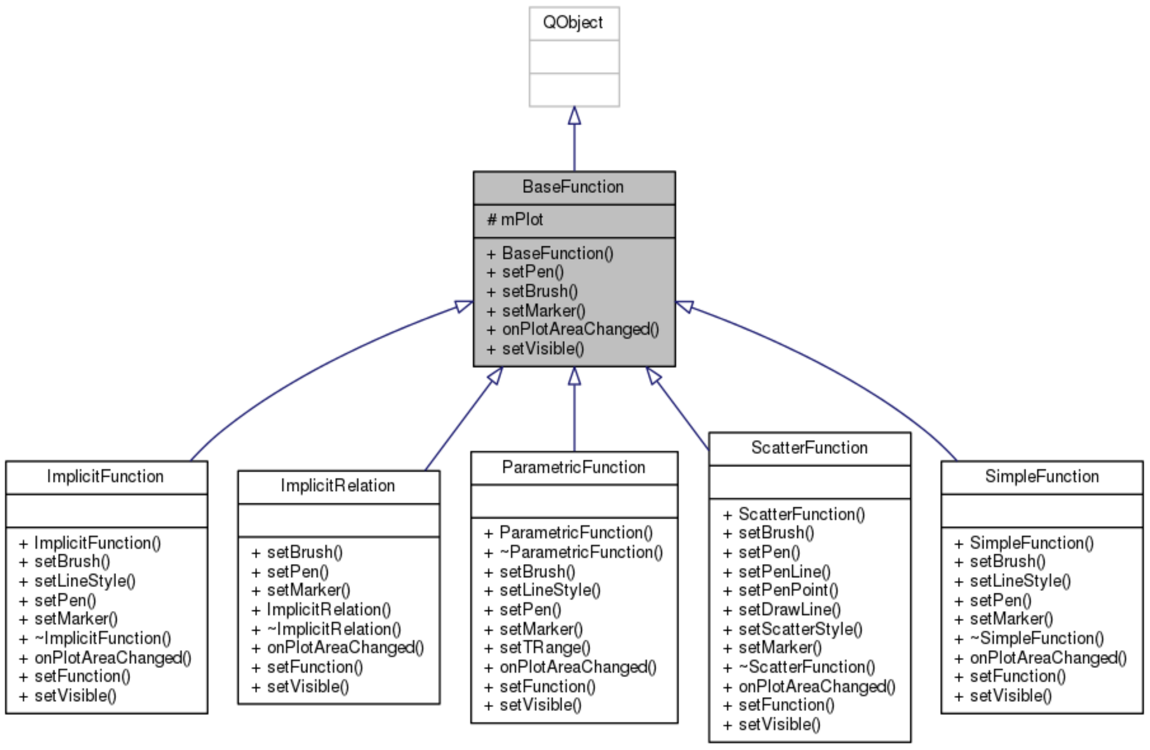

**Fig 4:** Base Function Class Diagram

### iv. Plotter Module:

We have made completely our own custom plotter for handling 3D cases using OpenGL, and extended the QCustomPlot widget for 2D plots.
*OpenGL based 3D Plotter*: We have a custom 3D Plotter, that takes in a heightmap, and a colormap, and renders the corresponding plot. It is rendered using the scene graph in QtQuick. The user can zoom in/out, pan and rotate the graph. The range for the graph changes dynamically. By not relying on 3rd party
libraries, we have made our plotter completely extendable for future open source use.

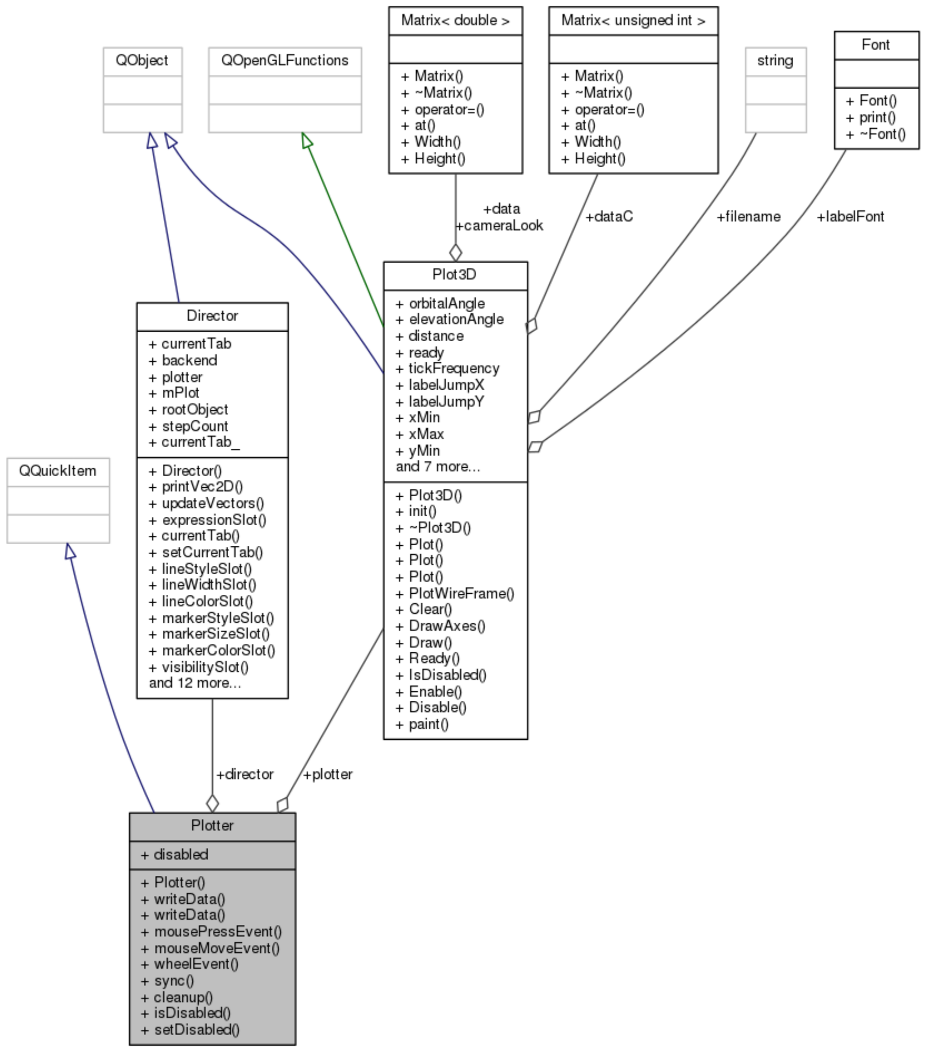

**Fig 5:** Plotter Class Diagram

*2D Potter:* We have extended the opensource QCustomPlotter for 2D cases, by adding event handling mechanisms, and additional classes for handling different types of functions.
For handling implicit cases, we utilise a novel method of Marching Squares.

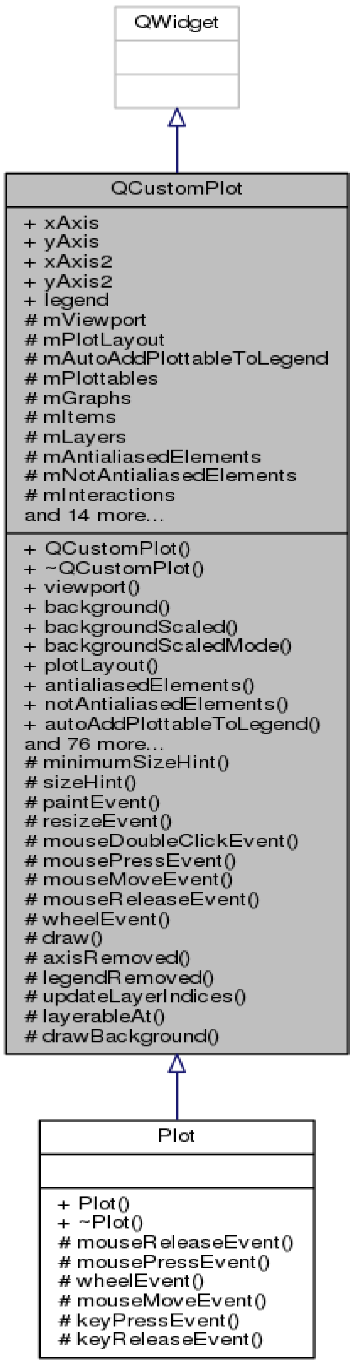

**Fig 6:** Plot Class Diagram

We adopt a quad tree based approach in case of inequalities. Here the entire grid is recursively divided into smaller squares. When the signs at all the corners of the square are the same, we know that it belongs to a specific region, otherwise, we divide the regions further.

#### v. Controller/Director Module:

This module contains the controls the flow of data across different modules. It maintains an Expression directory, which is updated as and when we add/delete/update. It also invokes the evaluate method call that returns a height map, which is used by the plotter to draw to the GUI.

Saving Module: In case of 3D, we use opensource GL2PS for saving vector graphics. It supports ".svg" format. Apart from that we used our own BMP file generator code, for saving 3D plots in raster BMP format.

### c. Cohesion and Coupling

By dividing our entire system into separate smaller and independent modules, we ensure high cohesion. For example, the expression class provides all information about an expression including its type, its range, and the corresponding OpenCL kernel. The Expression class thus caters only to expression properties.
Another instance is the usage of the BaseFunction abstract class for 2D functions, which has as its different types of children, the different function classes like Implicit, Explicit, Parametric etc. Thus, this class implements dynamic polymorphism effectively, and caters to the computation of 2D functions.
Each module is independent of each other and has few number of input and output parameters. The Parser class takes in a string and returns the C++ string, which is used by the Expression class, along with ranges, to return the input and heightmap vectors.
Coupling is directly proportional to number of input and output data parameters, thus by keeping the number of parameters low, we ensure lower coupling, which is always desirable.

### d. Reprogrammibility and Extendibility

The different modules and the Graphical User Interface can be extended to include additional functionality. This functionality includes

- Extending the parser to add more functions. This can be easily added by adding the corresponding functions to the constants.h and functions.h file.
- The abstract class Base Functions allows extensibility by adding derived classes as necessary for the different mathematical functions.
- The OpenCL kernel is cross platform, and thus can be easily extended to another platform.
- The GUI is designed in QTQuick, which is also cross-platform and thus requires minimum programming to extend to another platform.

- Also, a distinguishing feature of the GUI is that it is extended to work with both traditional desktop PCs and also touch interfaces (like tablets, smartphones). This ensures minimum reprogammability and easily extendibility.

## e. Use of Open Source libraries

The motivation for this project is to go "open source". The project thus has only open source external dependencies, and even in that case, we minimize the dependencies on external libraries as much as possible.
*Parser*: We use our **own** parser, which returns an equivalent C++ string from an infix string.
*OpenCl based Evaluation kernel*: This is also custom designed and relies only on OpenCL, which is a platform for parallel programming.
*Plotter 3D*: The plotter for 3D case is also custom designed in OpenGL, and rendered using scene graph in QtQuick
*Plotter 2D*: We have extended the QCustomPlot to include handling events, and also additional cases like shading and contour.
*Saving*: We have our own custom code for saving .bmp (raster ) file, though we rely on QQustomPlot for 2D saving purposes and GL2PS  for 3D saving.

# Chapter 3: Application Test Plan

## a. Introduction

### i. Description of this Test Plan

This part of the file is Test Plan for the *Graphi* application. It describes the testing strategy and approach to validate the quality of this application. It also contains various resources required for the successful completion of this project.
The document has been created in compliance with IEEE Standard 829.

### ii. Staffing and Timeline

Unit Testing is done by the developer himself and by a separate testing team. Manual test files are generated testing each aspect of the program (white box testing).  This done comprehensively during development, and immediately a day after the first draft of the module.
The integration testing shall begin after the modules are integrated one-by-one. This process shall take a week to complete. It shall be done by the programmers themselves. The Interface testing will happen periodically, as each separate feature is introduced. Interface revision happens based on feedback and user experience. Finally, the software shall undergo Alpha testing, wherein the installer will be circulated to everyone in the hall who is interested in testing the software. The software shall be hosted on an online repository, where the bugs would be reported and corrected. This will continue for at least 2 days.

### iii. Features to be tested

The features which are mentioned in user manual are to be tested. Implemented features to be tested:

- Parser
- Evaluator
- 2D Plotter
  - Parametric
  - Implicit

    - Explicit
- 3D Plotter
- Settings
    - Line Settings
    - Marker Settings
    - Brush settings
    - Graph Settings
- Data Series Visualization
- Graph Interactions
- Saving
- User Interface

#### iv. Pass/Fail Criteria

The output on the UI shall be compared with the output by the testing team, manually.
The test shall be termed as Pass, if both the above operations give the same result for all the valid commands in the "**Graphi**" application.

#### v. Test deliverables

Test Plan
List of test cases (Expected output and observed output for a selected set of test inputs)

### b. Stages of Testing

#### i. Unit Testing

All the different functions and features were tested by the developer himself and also by a testing team. Unit testing and error handling ensures high cohesion and that each module is independent of another, and thus can be independently tested and worked upon. Unit testing ensures that all errors are stopped in the pipeline at this stage only. Unit testing was done comprehensively by manually down all possible equivalence classes of inputs possible to a module.

#### ii. Integration Testing

We also tested the integration of 2 or more modules. For instance, integration of the parser and the Expression Evaluator was tested by writing a testing program that generated test files by taking input as the infix text string and the output as the vector values given at specific

ranges.
Integration testing ensured that subsequent modules were coupled with least effort, and flawlessly.

### iii. User Interface Testing

The user interface was tested. The windows were resized, minimized and maximized. The graph interactions were tested moving the graphs and changing its color and size. The zoom in and zoom out functions were tested
for the graph.

### iv. System Testing (Black Box)

The software was released to the students of the hall in the form of an installer with the user manual and the installation manual.
The software was hosted on an online Git repository with restricted access, where the interested students could access the latest version of the software.
Issues were reported by the individuals and were corrected by the programmers.

## c. Test cases

| Input string | Expected output | Actual output | Pass/Fail |
|---|---|---|---|
| **sin((x^2)+(y^e))pi** | sin(x^2+y^2.71)*3.14 | sin(x^2+y^2.71)*3.14 | Pass |
| **sin(xe(ye(tan(x)epi)))** | sin(x*2.71*(y*2.71*(tan(x)*2.71*3.14))) | sin(x*2.71*(y*2.71*(tan(x)*2.71*3.14))) | Pass |
| **x^sin(y+(4/5*a))** | pow(x,sin((y+((4/5)*a)))) | pow(x,sin((y+((4/5)*a)))) | Pass |
| **ee** | (2.71^2) | (2.71^2) | Pass |
| **e$www** | 2.71$(w^3) | 2.71$(w^3) | Pass |
| **pii** | 3.14*i | 3.14*i | Pass |
| **sin(pi..344)** | sin(3.14..344) | sin(3.14..344) | Pass |
| **cos(tan(x+y))+(78*tan(x^sin(cos(d))** | (cos(tan((x+y)))+(78*tan(pow(x,sin(cos(d)))))) | (cos(tan((x+y)))+(78*tan(pow(x,sin(cos(d)))))) | Pass |
| **2*sin(x)+54*cos(e^(23+x))** | ((2*sin(x))+(54*cos(pow(e,(23+x))))) | ((2*sin(x))+(54*cos(pow(e,(23+x))))) | Pass |
| **e^6*pi*pi*p*x^2*y^2** | (2.71^6)*(3.14^2)*p*(x^2)*y | (2.71^6)*(3.14^2)*p*(x^2)*y | Pass |

### i. Parser

**Some Snippets:**

- pi .... 3.14
- e .... 2.71
- pi^x .... 3.14^x
- e^x .... 2.71^x

- e^pi .... 2.71^3.14
- (pi*pi^e^pi) .... (3.14*3.14^2.71^3.14)
- pi+pi*pipi .... 3.14+3.14*(3.14^2)
- pi+e/e/pi .... 3.14+2.71/2.71/3.14
- pi/pi .... 3.14/3.14
- e^(pie) .... 2.71^(3.14*2.71)
- pi-e+2.334pi .... 3.14-2.71+2.334*3.14
- sin(pix) .... sin(3.14*x)
- sin(e*pi) .... sin(2.71*3.14)
- sin(epi*xy) .... sin(2.71*3.14*x*y)
- sin(etan(pixy)) .... sin(2.71*tan(3.14*x*y)) pix .... 3.14*x
- pisin(ex) .... 3.14*sin(2.71*x)
- sin(x):sin(x)
- cos(x):cos(x)
- tan(x):tan(x)
- sin(x)*cos(y):(sin(x)*cos(y))
- cos(x)*sin(y)/tan(z):((cos(x)*sin(y))/tan(z))
- 345*sin(x)/cos(x):((345*sin(x))/cos(x))
- 123-sin(x):(123-sin(x))
- 145+tan(x)-sin(y)+(45*cos(x)):(((145+tan(x))-sin(y))+(45*cos(x)))
- sin(x*y+z):sin(((x*y)+z))
- cos(x^y+s^f):cos((pow(x,y)+pow(s,f)))
- tan(a/3^w^(d+e)):tan((a/pow(3,pow(w,(d+e)))))
- sin(2):sin(2)

## ii. Evaluator

| Input String | Input | | | Expected Output | Actual Output | Pass/Fail |
|---|---|---|---|---|---|---|
| | Lower Limit | Upper Limit | Steps | | | |
| (x+2)/(x^2-x-6) | 83.7298 | 90.4823 | 6.24942 | 0.012387 0.011497 | 0.012387 0.011497 | Pass |
| sin(1/x) | -15.784 | 85.8268 | 1.1348 | -0.0633131<br>-0.0682104<br>-0.073928… | -0.0633131<br>-0.0682104<br>-0.073928… | Pass |
| xcos(abs(x^3-x)) | -2.92631 | 84.7735 | 1.80158 | 2.89707<br>-1.07513<br>0.631821… | 2.89707<br>-1.07513<br>0.631821… | Pass |
| x^3.231*4 | -31.2505 | 93.1227 | 16.1197 | -nan<br>-nan<br>3.85687<br>38597.9<br>329644<br>1.18306e+06<br>2.94875e+06<br>6.00494e+06 | -nan<br>-nan<br>3.85687<br>38597.9<br>329644<br>1.18306e+06<br>2.94875e+06<br>6.00494e+06 | Pass |

**Some snippets:**

- sin(-x)+-cos(x):68.0589 70.0857 1.54933 0.378237 -1.35426
- x^3.231*4:-31.2505 93.1227 16.1197 -nan -nan 3.85687 38597.9 329644 1.18306e+06 2.94875e+06 6.00494e+06
- arcsin(cosec(x)):-17.7151 86.1838 1.69255 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
- cos(pi-3.14)x^3:73.8991 85.3474 17.4134 403568
- e^xsin(x)+ln(e^3x):-39.0495 78.3527 11.2435 nan nan nan nan -126.569 -2.83937e+07 -2.97942e+11 1.54474e+17 7.52776e+21 -6.19018e+26 -6.71155e+31
- sin(x+pi5)ln(pisin(x)):-99.0762 -61.535 3.04594 -1.13025 nan -1.04518 nan -0.891075 nan -0.682336 nan -0.440282 nan -0.194003 nan 0.0157419
- sin(sqrt(tan(x))):36.2681 68.5659 8.44329 -nan 0.810548 -nan -nan
- arctan(e^sin(x+2.42)cos(x-2.42)):-72.1798 -68.7449 18.9681 0.36541
  pie^epiepix:86.2923 92.4156 8.23565 110196
- ln(sin(x^x)):-16.7896 0.794766 7.12782 -nan -nan -nan

abs(x^2-2)+abs(x+2):83.6763 93.0474 16.8683 7085.4